1. What is React?

* React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies. It is basically used for making single page applications.
* Note- In multipage application for every request of different app pages that user makes to server, a separate html,css and js files are send by the server but in single page applications only one time request is send to server and server sends the html, css and js files only once after that for every different page of app JS file takes over command and all HTML and CSS components are same and changes are done on same page only. Same page is populate and reload using APIs. That is why single page apps made from React.js like library are faster than multi page normal apps.
* It is useful to create applications with minimum code of js and that too in organised manner (Not like other frameworks of js).

1. Getting Started:

* Installation

1. React dev tools (Google chrome extension)
2. VS code
3. Node.js
4. NPM

* VS code extensions

1. Thunder client
2. Live server
3. auto rename tag
4. ES7/ react/redux/ graphQL
5. Creating first react app:

* Open any folder
* Use command – **npx create-react-app app\_name**
* Note- We can also use command like **npm create-react-app app\_**name but this is used for installing react app globally, and this will occupy more space and ram of your system therefore we will use npx which deals with take-use-return policy.
* Move to that app\_name folder by command – **cd app\_name**
* Write command **– npm start**
* Now your react app will start at localhost:3000
* Don’t change anything in index.html file, we can just add bootstrap or js files inside it.
* Do changes in src/app.js file for showing changes on app.
* Our react app use function based components(Earlier it uses class based components)

1. Folder structure of react app:

* It contains major 2 folders that needs to be deal with. 1st is public folder and 2nd is src folder
* Public folder contains index.html file in which we can add bootstrap and js
* Src folder is the main folder that contains app.js in which we need to change things according to our project. App.js is directly connected with index.html and any changes done in app.js will get render on webpage directly.
* Src folder also contains component folder that have all components.js files for our react app.
* <React.StrictMode> in index.js file ensures that you write a very good code without syntax error so that overall file is bug free. For ex- if strictMode is not present than if you just write a=10, it will run on some consoles by auto detecting the type of variable, but good way to write this inside js file is -> let a=10; Strict Mode will ensure these things to not happen.

Note:

* Difference between **let** and **var** keywords in JavaScript:
  + For example : if we write in js console-

**var a=’reet’;**

**var a=24**

**console.log(a)**

**This will change value of variable a from ‘reet’ to 24, but ideally it should give error, because a is already been defined.**

**Therefore in modern JS versions let keyword is defined**

If we write

**let a=’reet’;**

**let a=90;**

**Console.log(a);**

This will give error that is, ‘a’ is already defined

* Const keyword:
  + The variable that is defined using const keyword is fixed and cannot be changed (immutable).

1. JSX:

* Const element=<h1>Hello, world!</h1>;
* This funny tag syntax is neither a string nor HTML.
* It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.
* It can be wrote in app.js file under function ()-

For ex- const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>;

* It is 90% html code that is written in app.js file but have some changes which must be done like:

class replace by 🡪 className

for replaced by 🡪 htmlFor

unterminated html tags like **<img src=””>** should be replaced by **<img src=””/>**

* JSX only returns **one code segment inside one function** in app.js that means if we want to create more divs and write some more html code we have to bind them to return full JSX only one time. We can do this by embedding full code inside **<> </> braces or JSX fragment.**
* Example- APP.js
* import logo from './logo.svg';
* import './App.css';
* function App() {
* return (
* <>               {/\*JSX fragment opening\*/}
* <h1>Myself Tomar</h1>
* <div className="App">
* <header className="App-header">
* <img src={logo} className="App-logo" alt="logo" />
* <p>
* Edit <code>src/App.js</code> and save to reload.
* </p>
* <a
* className="App-link"
* href="https://reactjs.org"
* target="\_blank"
* rel="noopener noreferrer"
* >
* Learn React
* </a>
* </header>
* </div>
* </>
* );
* }
* export default App;

1. Adding Bootstrap to react:

* Just go to bootstrap website and search started template. From starter template copy the code of adding JS and Adding CSS to your index.html in body and head respectively.
* Now, you can use bootstrap components by writing code inside app.js in function as a JSX code inside JSX fragments <></>.

1. Making **components** in react app:

* To create components, go to src folder and create one components folder.
* Create any components inside components, for example- **Navbar.js** (Note: The **component name** should always starts with **capital letter**)
* Now we need to either write code for **react function based components** inside our component file (Navbar.js) or 2nd option is to call **rfc.**
* In **VS code** if you install **ES7 React/Redux** extension, it provides you to use shortcuts like rfc, that will call the code of react function based component.
* Ass, we do in App.js file by writing more than one div or sections inside JSX<></>. Here also, to write more than one div we have to write everything in <></> brackets because the function based component returns only one thing, that’s why we have to write everything inside these brackets. But, if only single thing is need to be return, than we don’t need them inside the file.
* For example- Just **type rfc inside Navbar.js in VS code**, and ES7 extension will provide you the following code-
* import React from 'react'
* export default function Navbar() {
* return (
* <div>
* </div>
* )
* Now write the **code of Navbar**, or just copy paste from Bootstrap and **paste it inside <div></div> of Navbar.js file**
* Now file will look like-

import React from "react";

export default function Navbar() {

  return (

<>

    <div>

      <nav className="navbar navbar-expand-lg navbar-light bg-light">

        <a className="navbar-brand" href="/">

          textup

        </a>

        <button

          className="navbar-toggler"

          type="button"

          data-toggle="collapse"

          data-target="#navbarSupportedContent"

          aria-controls="navbarSupportedContent"

          aria-expanded="false"

          aria-label="Toggle navigation"

        >

          <span className="navbar-toggler-icon"></span>

        </button>

        <div className="collapse navbar-collapse" id="navbarSupportedContent">

          <ul className="navbar-nav mr-auto">

            <li className="nav-item active">

              <a className="nav-link" href="/">

                Home <span className="sr-only">(current)</span>

              </a>

            </li>

          <form className="form-inline my-2 my-lg-0">

            <input

              className="form-control mr-sm-2"

              type="search"

              placeholder="Search"

              aria-label="Search"

            />

            <button

              className="btn btn-outline-success my-2 my-sm-0"

              type="submit"

            >

              Search

            </button>

          </form>

        </div>

      </nav>

    </div>

</>

  );

}

* Now you just need to call this Navbar component from your App.js file to use it anywhere in any applications. (Note: To call any component file in App.js, write its name inside ‘</>’ tag before ‘/’ inside JSX fragments i.e <></>.

**App.js –**

import './App.css';

import Navbar from './components/Navbar';

function App() {

  return (

    <>

      <Navbar/>

    </>

  );

}

export default App;

* **Main application of using Components in React (Components based architecture):**

**In React we make components to reuse them anytime anywhere in the app and the 2nd use of component is to make sure that your App.js is clean and easily understandable.**

1. **Props** in React:
2. function Welcome(props) {
3. return <h1>{props.name}</h1>;
4. }

* This function is a valid React component because it accepts a single **“props”** (which **stands** **for properties**) object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions.
* In our previous example we can set props in <Navbar/> component.

Syntax: App.js File

import './App.css';

import Navbar from './components/Navbar';

function App() {

  return (

    <>

     <Navbar title="TextUp" about="About TextUp"/> //setting properties/props

    </>

  );

}

export default App;

* This means we just need to write the name of prop with their respected values inside their tags (here <Navbar/>)
* Now, go inside the component file i.e Navbar.js inside components folder of your app and just write **props** inside brackets of function. (It will act as its parameter)

export default function Navbar(props) {

* Now, instead of writing values in your code in components file. Just call the prop name inside curly braces i.e {}.

In our Navbar.js

<a className="navbar-brand" href="/">

          {props.title}

</a>

* Similarly do it for about tag also
* <a className="nav-link" href="/">
* {props.about}<span className="sr-only">(current)</span>
* </a>
* Our overall file Navbar.js will now look like: (Note: please match and compare it with previous Navbar.js, you will only see the above changes)

import React from "react";

export default function Navbar(props) {

  return (

    <div>

      <nav className="navbar navbar-expand-lg navbar-light bg-light">

        <a className="navbar-brand" href="/">

          {props.title}

        </a>

        <button

          className="navbar-toggler"

          type="button"

          data-toggle="collapse"

          data-target="#navbarSupportedContent"

          aria-controls="navbarSupportedContent"

          aria-expanded="false"

          aria-label="Toggle navigation"

        >

          <span className="navbar-toggler-icon"></span>

        </button>

        <div className="collapse navbar-collapse" id="navbarSupportedContent">

          <ul className="navbar-nav mr-auto">

            <li className="nav-item active">

              <a className="nav-link" href="/">

                {props.about}<span className="sr-only">(current)</span>

              </a>

            </li>

            <li className="nav-item dropdown">

              <a

                className="nav-link dropdown-toggle"

                href="/"

                id="navbarDropdown"

                role="button"

                data-toggle="dropdown"

                aria-haspopup="true"

                aria-expanded="false"

              >

                Dropdown

              </a>

              <div className="dropdown-menu" aria-labelledby="navbarDropdown">

                <a className="dropdown-item" href="/">

                  Action

                </a>

                <a className="dropdown-item" href="/">

                  Another action

                </a>

              </div>

            </li>

          </ul>

          <form className="form-inline my-2 my-lg-0">

            <input

              className="form-control mr-sm-2"

              type="search"

              placeholder="Search"

              aria-label="Search"

            />

            <button

              className="btn btn-outline-success my-2 my-sm-0"

              type="submit"

            >

              Search

            </button>

          </form>

        </div>

      </nav>

    </div>

  );

}

9) PropType in React:

* Sometime user makes mistakes while passing props/parameters inside component tags in App.js file.

For example- In our previous case of TextUp app, the Navbar.js components have 2 props i.e title and about prop. We can set the data type of both these parameters, because here the title should be a string, but suppose if in future the user enters a number instead of string inside title prop, than it becomes a mistake. These mistakes are hard to detect, therefore we reduce our risk by setting PropType in component file.

* Firstly import proptypes inside your Navbar.js file. (use ‘impt’ shortcut in VS Code)
* import PropTypes from 'prop-types'

Write the above command at top of the Navbar.js file

* Now, in Navbar.js, add this thing at the end of the file:
* Navbar.propTypes= {
* title: PropTypes.string.isRequired,
* about: PropTypes.string
* }

**This will ensure that our props title and about are string types and if user tries to give any other data type inside them, than it will show error in console.**

* We also have some other options like:
  + **defaultProps :** This will ensure that if user forget to set values of any prop while calling the component, than the default prop will be called.
  + Syntax: Add this at end of component file Navbar.js
* Navbar.defaultProps= {
* title: 'Set title here',
* about: 'Set About app here'
* }

* Now, our overall Navbar.js file will look like:

import React from "react";

import PropTypes from 'prop-types'

export default function Navbar(props) {

  return (

    <>

      <nav className="navbar navbar-expand-lg navbar-light bg-light">

        <a className="navbar-brand" href="/">

          {props.title}

        </a>

        <button

          className="navbar-toggler"

          type="button"

          data-toggle="collapse"

          data-target="#navbarSupportedContent"

          aria-controls="navbarSupportedContent"

          aria-expanded="false"

          aria-label="Toggle navigation"

        >

          <span className="navbar-toggler-icon"></span>

        </button>

        <div className="collapse navbar-collapse" id="navbarSupportedContent">

          <ul className="navbar-nav mr-auto">

            <li className="nav-item active">

              <a className="nav-link" href="/">

                {props.about}<span className="sr-only">(current)</span>

              </a>

            </li>

            <li className="nav-item dropdown">

              <a

                className="nav-link dropdown-toggle"

                href="/"

                id="navbarDropdown"

                role="button"

                data-toggle="dropdown"

                aria-haspopup="true"

                aria-expanded="false"

              >

                Dropdown

              </a>

              <div className="dropdown-menu" aria-labelledby="navbarDropdown">

                <a className="dropdown-item" href="/">

                  Action

                </a>

                <a className="dropdown-item" href="/">

                  Another action

                </a>

              </div>

            </li>

          </ul>

          <form className="form-inline my-2 my-lg-0">

            <input

              className="form-control mr-sm-2"

              type="search"

              placeholder="Search"

              aria-label="Search"

            />

            <button

              className="btn btn-outline-success my-2 my-sm-0"

              type="submit"

            >

              Search

            </button>

          </form>

        </div>

      </nav>

    </>

  );

}

Navbar.propTypes= {

    title: PropTypes.string.isRequired,

    about: PropTypes.string

 }

Navbar.defaultProps= {

    title: 'Set title here',

    about: 'Set About app here'

 }

10) Hooks in React (Ex- UseState):

* Hooks are new addition in React 16.8. They let you use State and other React features without writing a class.
* Advantage is – we can directly make changes in a defined variable with help of its respective function without creating a class for it.
* Hooks especially the UseState hook, can be said as a pair of current state/value/variable and function that changes the current state.
* The initial value of state is defined inside brackets of useState, as shown below:

const [count, setCount] = useState(0);

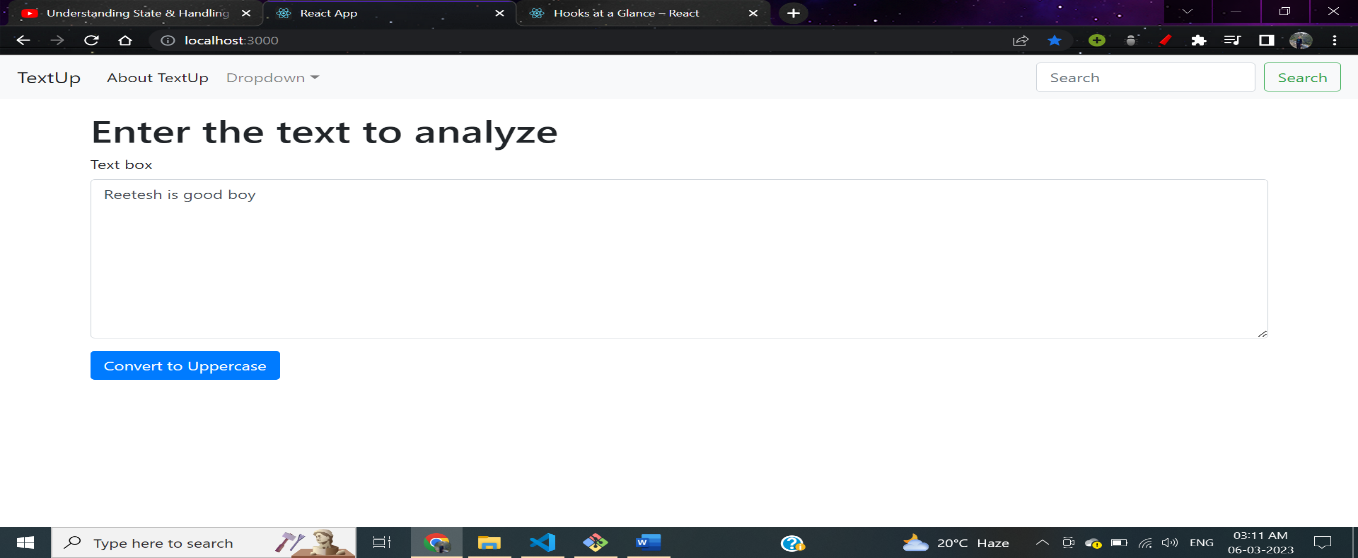
* Here the initial value of count is 0.
* If we want to change the count variable using setCount, we can write like this:

<button onClick={() => setCount(count + 1)}>

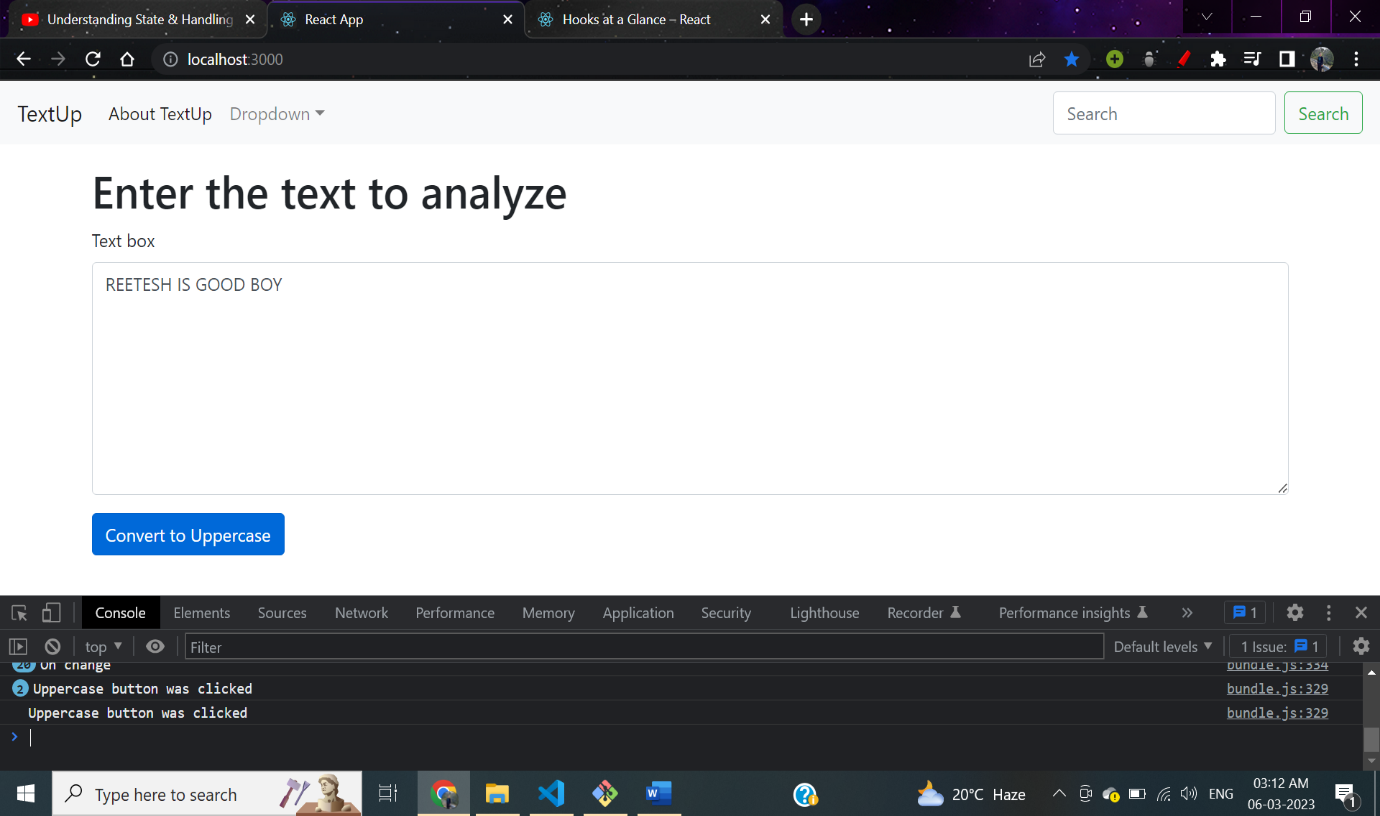
* Ex-
  + **State Hook**
* import React, { useState } from 'react';
* function Example() {
* // Declare a new state variable, which we'll call "count"
* const [count, setCount] = useState(0);
* return (
* <div>
* <p>You clicked {count} times</p>
* <button onClick={() => setCount(count + 1)}>
* Click me
* </button>
* </div>
* );
* }
* This example renders a counter. When you click the button, it increments the value.
* Here, useState is a Hook. We call it inside a function component to add some local state to it.
* React preserve this state between re-renders.
* **useState returns a pair: the current state value and a function that lets you update it.**
* You can call this function from an event handler or somewhere else. It’s similar to this.setState in a class, except it doesn’t merge the old and new state together.
* **The only argument to useState is the initial state. In the example above, it is 0 because our counter starts from zero.**
* But what is a Hook?
  + Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes. (We [don’t recommend](https://reactjs.org/docs/hooks-intro.html#gradual-adoption-strategy) rewriting your existing components overnight but you can start using Hooks in the new ones if you’d like.)
  + React provides a few built-in Hooks like useState. You can also create your own Hooks to reuse stateful behavior between different components. We’ll look at the built-in Hooks first.

**>> Example of Hook in Our application of TEXTUP:**

* In our Example of TextForm.js Component, we are using useState hook to update the initial and current state of text value inside the textArea, that we have created in our application.
* We are changing the Text of our textarea to UpperCase. Here is the code:
* import React, {useState} from "react";
* export default function TextForm(props) {
* const handleUpClick=()=>{
* console.log("Uppercase button was clicked");
* let newText=text.toUpperCase();
* setText(newText);
* }
* const handleOnChange=(event)=>{
* console.log("On change");
* setText(event.target.value);
* }
* const [text, setText] = useState('Enter text here');
* return (
* <div>
* <h1>{props.heading}</h1>
* <div className="form-group">
* <label htmlFor="myBox">Text box</label>
* <textarea
* className="form-control"
* value={text}
* onChange={handleOnChange}
* id="myBox"
* rows="8"
* ></textarea>
* </div>
* <button className="btn btn-primary" onClick={handleUpClick}>Convert to Uppercase</button>
* </div>
* );
* }
* Here, we are using useState hook and defined it as pair :
* const [text, setText] = useState('Enter text here');
* The text is the variable that represent the value of Textarea in application and setText is the function that is been used to change the state of text wherever required.
* We are changing state of text inside handleOnChange and handleUpclick functions. Whenever user press the change to Uppercase button in app, the value of textarea changes to UpperCase.
* Screenshots of working application : Pic-1) Writing text in textarea



Pic-2) Converting the text into uppercase by clicking the convert to Uppercase button



**11) Routers (For Single Page applications):**

* React Router enables **“client side routing”.** In traditional websites, the browser requests a document from a web server, downloads and evaluates CSS and JavaScript assets, and renders the HTML sent from the server. When the user clicks a link, it starts the process all over again for a new page.
* Client side routing allows your app to update the URL from a link click without making another request for another document from the server. Instead, your app can immediately render some new UI and make data requests with ‘fetch’ to update the page with new information.
* This **enables faster user experiences because the browser doesn’t need to request an entirely new document** or re-evaluate CSS and JS assets for the next page. It also enables more dynamic user experiences with things like animation.
* Client side routing is enabled by creating a ‘Router’ and linking/submitting to pages with ‘Link’ and ‘<Form>’.
* The following are the steps to do basic routing:
  + First install **create-react-app** and make a new project with it.
  + In terminal write command-

npm install react-router-dom

* + After installing react-router-dom, we can make use of tags like Routes, Link, Switch, etc.
  + In our example textUP app, we have 2 “pages” handled by the router: a home page and an about page. As you click around on the different <Link>s, the router renders the matching <Route>.
  + Note: Behind the scenes a **<Link> renders an <a>** with a real href, so people using the keyboard for navigation or screen readers will still be able to use this app.
  + Now, in the **app.js** file**, import necessary things such as Router, Routes, Woute, Link from react-router-dom as shown in example code of our app TextUp.**
  + After importing things, its time to use them, here is the syntax, to call each page using <route>:
* <Router>
* <Navbar   mode={mode} title="TextUp" about="About TextUp" toggleMode={toggleMode}/>
* <div className="container my-3" >
* <Routes>
* {/\* We are using "exact" keyword before path because, if we don't put it than also our app will run but in big apps where we have lot of routers/pages of similar names, React uses partial matching, and because of it rendering pages with similar name will be confusing \*/}
* {/\* for example: route1/Component1 =    /users \*/}
* {/\* route2/Component2 =    /users/home \*/}
* {/\* If we don't use exact keyword, than while calling component2, React will do partial matching and call just /users i.e component 1 \*/}
* <Route exact path="/about" element={<About mode={mode} />}/>
* <Route exact path="/" element={<TextForm  mode={mode} heading="Enter the text to analyze below"/>}/>
* </Routes>
* </div>

      </Router>

* + **Full example code from TextUP app-**
* import './App.css';
* import About from './components/About';
* import Navbar from './components/Navbar';
* import TextForm from './components/TextForm';
* import React, { useState } from 'react';
* import {
* BrowserRouter as Router,
* // Switch,    //It was used in previous versions of react-route-dom such as version 5.0, but in latest version 6.8.2, we use routes
* Routes,
* Route,
* Link
* } from "react-router-dom";
* function App() {
* const [mode,setMode]=useState('light');  //Setting dark/loght mode
* const toggleMode =()=>{
* if(mode==='light'){
* setMode('dark');
* document.body.style.backgroundColor='#042743';
* }
* else{
* setMode('light');
* document.body.style.backgroundColor='white';
* }
* }
* return (
* <>
* <Router>
* <Navbar   mode={mode} title="TextUp" about="About TextUp" toggleMode={toggleMode}/>
* <div className="container my-3" >
* <Routes>
* {/\* We are using "exact" keyword before path because, if we don't put it than also our app will run but in big apps where we have lot of routers/pages of similar names, React uses partial matching, and because of it rendering pages with similar name will be confusing \*/}
* {/\* for example: route1/Component1 =    /users \*/}
* {/\* route2/Component2 =    /users/home \*/}
* {/\* If we don't use exact keyword, than while calling component2, React will do partial matching and call just /users i.e component 1 \*/}
* <Route exact path="/about" element={<About mode={mode} />}/>
* <Route exact path="/" element={<TextForm  mode={mode} heading="Enter the text to analyze below"/>}/>
* </Routes>
* </div>
* </Router>
* </>
* );
* }
* export default App;
  + That’s all, after these steps, our single page app will be ready to render all pages, without reloading anything.